软 件 设 计 方 案

第一章、用户界面设计规范

用户界面：又称人机界面，实现用户与计算机之间的通信，以控制计算机或进行用户与计算机之间的数据传送的系统部件。

GUI：即图形用户界面，一种可视化的用户界面，它使用图形界面代替正文界面。

本系统坚持图形用户界面（GUI）设计原则，界面直观、对用户透明。用户接触软件后对界面上对应的功能一目了然、不需要多少培训就可以方便地使用本应用系统。

1、界面设计介绍

界面设计是为了满足软件专业化标准化的需求而产生的对软件的使用界面进行美化优化规范化的设计分支。

1）软件启动封面设计

应使软件启动封面最终为高清晰度的图像，选用的色彩不宜超过 256 色，大小多为主流显示器分辨率的 1/6 大。启动封面上应该醒目地标注制作或支持的公司标志、产品商标、软件名称、版本号、网址、版权声明、序列号等信息，以树立软件形象，方便使用者或购买者在软件启动的时候得到提示。插图宜使用具有独立版权的、象征性强的、识别性高的、视觉传达效果好的图形，若使用摄影也应该进行数位处理，以形成该软件的个性化特征。如果是系列软件还将考虑整体设计的统一和延续性。

2）软件框架设计

软件的框架设计要复杂得多。软件框架设计应该简洁明快，尽量少用无谓的装饰，应该考虑节省屏幕空间，各种分辨率的大小，缩放时的状态和原则，并且为将来设计的按钮、菜单、标签、滚动条及状态栏预留位置。设计中将整体色彩组合进行合理搭配，将软件商标放在显著位置，主菜单应放在左边或上边，滚动条放在右边，状态栏放在下边，以符合视觉流程和用户使用心理。

3）软件按钮设计

软件按钮设计应该具有交互性，即应该有 3 到 6 种状态效果：点击前鼠标未放在上面时的状态；鼠标放在上面但未点击的状态；点击时状态；点击后鼠标未放在上面时的状态；不能点击时状态；独立自动变化的状态。按钮应具备简洁的图示效果，名称易懂，用词准确，能望文知意最好，让使用者产生功能关联反应，群组内按钮应该风格统一，功能差异大的按钮应该有所区别。

4）软件面板设计

软件面板设计应该具有缩放功能，面板应该对功能区间划分清晰，应该和对话框、弹出框等风格匹配，尽量节省空间，切换方便。

5）菜单设计

菜单设计一般有选中状态和未选中状态，左边应为名称，右边应为快捷键。如

果有下级菜单应该有下级箭头符号，不同功能区间应该用线条分割。 对与进行的操作无关的菜单要用屏蔽的方式加以处理，如果采用动态加载方式，即只有需要的菜单才显示最好。主菜单的宽度要接近，字数不应多于四个，每个菜单的字数能相同最好。 主菜单数目不应太多，最好为单排布置。

6）标签设计

标签设计应该注意转角部分的变化，状态可参考按钮。

7）图标设计

图标设计色彩不宜超过 64 色，大小为 16x16、32x32 两种，应该加以着重考虑视觉冲击力，它需要在很小的范围表现出软件的内涵，在设计时使用简单的颜色，利用眼睛对色彩和网点的空间混合效果，做出精彩图标。

8）滚动条及状态栏设计

滚动条主要是为了对区域性空间的固定大小中内容量的变换进行设计，应该有上下箭头，滚动标等，有些还有翻页标。状态栏是为了对软件当前状态的显示和提示。

9）安装过程设计

安装过程设计主要是将软件安装的过程进行美化，包括对软件功能进行图示化。

10）包装及商品化

最后软件产品的包装应该考虑保护好软件产品，功能的宣传融合于美观中，可以印刷部分产品介绍。

2、界面设计原则

1）易用性

（1）完成相同或相近功能的按钮用 Frame 框起来，常用按钮要支持快捷方式；

（2）完成同一功能或任务的元素放在集中位置，减少鼠标移动的距离；

（3）按功能将界面划分局域块，用 Frame 框括起来，并要有功能说明或标题；

（4）界面要支持键盘自动浏览按钮功能，即按 Tab 键的自动切换功能；

（5）同一界面上的控件数最好不要超过 10 个，多于 10 个时可以考虑使用分页界面显示；

（6）分页界面要支持在页面间的快捷切换，常用组合快捷键 Ctrl Tab；

（7）默认按钮要支持 Enter 及选操作，即按 Enter 后自动执行默认按钮对应操作；

（8）可写控件检测到非法输入后应给出说明并能自动获得焦点；

（9）Tab 键的顺序与控件排列顺序要一致，目前流行从上到下、从左到右的方式；

（10）复选框和选项框要有默认选项，按选择机率的高低而先后排列，并支持 Tab 选择；

（11）界面空间较小时使用下拉框而不用选项框；

（12）选项数较少时使用选项框，相反使用下拉列表框；

（13）适当使用相关的专业术语，提倡使用通用性字眼。

2）规范性

通常界面设计都按 Windows 界面的规范来设计，即包含“菜单条、工具栏、工具厢、状态栏、滚动条、右键快捷菜单”的标准格式。小型软件一般不提供工具厢。

（1）菜单前的图标能直观地代表要完成的操作，常用菜单要有命令快捷方式 ；

（2）完成相同或相近功能的菜单用横线隔开放在同一位置，菜单深度一般要求最多控制在三层以内；

（3）相同或相近功能的工具栏放在一起，工具栏中的每一个按钮要有及时提示信息；

（4）系统常用的工具栏设置默认放置位置，工具栏的图标能直观地代表要完成的操作，一条工具栏的长度不能超出屏幕宽度；

（5）工具栏太多时可以考虑使用工具厢； 工具厢要具有可增减性，由用户自己根据需求定制，默认总宽度不要超过屏幕宽度的 1/5；

（6）状态条要能显示用户切实需要的信息，常用的有：目前的操作、系统状态、用户位置、用户信息、提示信息、错误信息等，高度以放置五好字为宜；

（7）滚动条的长度要根据显示信息的长度或宽度能及时变换，以利于用户了解显示信息的位置和百分比，并且宽度应比状态条的略窄；

（8）菜单和工具条要有清楚的界限，菜单要求凸出显示，这样在移走工具条时仍有立体感；

（9）菜单和状态条中通常使用五号字体。工具条一般比菜单要宽，但不要宽得太多，否则看起来很不协调；

（10）右键快捷菜单采用与菜单相同的准则。

3）合理性

屏幕对角线相交的位置是用户直视的地方，正上方四分之一处为易吸引用户注意力的位置，在放置窗体时要注意利用这两个位置。

（1）父窗体或主窗体的中心位置应该在对角线焦点附近；

（2）子窗体位置应该在主窗体的左上角或正中，多个子窗体弹出时应该依次向右下方偏移，以显示出窗体标题为宜；

（3）重要的命令按钮与使用较频繁的按钮要放在界面上注目的位置；

（4）与正在进行的操作无关的按钮应该加以屏蔽(Windows 中用灰色显示，没法使用该按钮) ；

（5）对可能造成数据无法恢复的操作必须提供确认信息，给用户放弃选择的机会。

4）美观与协调性

（1）按钮大小基本相近，且与界面的大小、空间要协调，忌用太长的名称；

（2）避免空旷的界面上放置很大的按钮，放置完控件后界面不应有很大的空缺位置；

（3）前景与背景色搭配合理协调，反差不宜太大，最好少用深色，常用色考虑使

用 Windows 界面色调；

（4）界面风格要保持一致，字的大小、颜色、字体要相同，除非是需要艺术处理或有特殊要求的地方；

（5）如果窗体支持最小化、最大化或放大时，窗体上的控件也要随着窗体而缩放；

（6）对于含有按钮的界面一般不应该支持缩放，即右上角只有关闭功能；

（7）通常父窗体支持缩放时，子窗体没有必要缩放。

5）界面一致性

在界面设计中应该保持界面的一致性。一致性既包括使用标准的控件,也指使用相同的信息表现方法，如在字体、标签风格、颜色、术语、显示错误信息等方面确保一致。

（1）显示信息一致性

① 标签提示：字体为不加粗、宋体、黑色、灰底或透明、无边框、右对齐、不带冒号、一般情况为五号；

② 日期：正常字体、宋体、白底黑字；

③ 对齐方法

左对齐：一般文字、单个数字、日期等

右对齐：数字、时间、日期加时间

④ 分辨率 800*600，增强色 16 色；

⑤ 字体缺省为宋体、五号、黑色；

⑥ 底色缺省为灰色。

这些信息的排列显示风格供参考，在同一软件中应当注意表现形式的一致性。

（2）布局合理化

应注意在一个窗口内部所有控件的布局和信息组织的艺术性，使得用户界面美观。布局不宜过于密集，也不能过于空旷，合理的利用空间。

在一个窗口中按 tab 键，移动顺序不能杂乱无章，先从上至下，再从左至右。一屏中首先应输入的和重要信息的控件在 tab 顺序中应当靠前,并放在窗口上较醒目的位置。布局力求简洁、有序、易于操作。

（3）鼠标与键盘对应

应用中的功能只用键盘也应当可以完成，即设计的应用中还应加入一些必要的按钮和菜单项。但是，许多鼠标的操作，如双击、拖动对象等，并不能简单地用键盘来模拟即可实现。例如在一个列表框中用鼠标单击其中一项表示选中该项内容，为了用键盘也能实现这一功能，必须在窗口中定义一个表示选中的按钮，以作为实现单击功能的替。又如在一个窗口中有两个数据窗口，可以用鼠标从一个数据窗口中将一项拖出然后放到另一个中，如果只用键盘，就应当在菜单中设置拷贝或移动的菜单项。

（4）快捷键

在菜单项中使用快捷键可以让使用键盘的用户操作得更快一些，在 Windows 及其应用软件中快捷键的使用大多是一致的。本系统中应用的快捷键在各个配置项上语义必须保持一致。

Ctrl-O 打开 Ctrl-Tab 下一窗口

Ctrl-S 保存 Ctrl-Esc 任务列表

Ctrl-C 拷贝 Ctrl-F4 关闭窗口
Ctrl-V 粘贴 Alt-F4 结束应用
Ctrl-D 删除 Alt-Tab 下一应用
Ctrl-X 剪切 Enter 缺省按钮/确认操作
Ctrl-I 插入 Esc 取消按钮/取消操作
Ctrl-H 帮助 Shift-F1 上下文相关帮助
Ctrl-P 打印
Ctrl-W 关闭

其它快捷键

其它快捷键使用汉语拼音的开头字母，不常用的可以没有快捷键。

6）向导

对于应用中某些部分的处理流程是固定的,用户必须按照指定的顺序输入操作信息，为了使用户操作得到必要的引示应该使用向导，使用户使用功能时比较轻松明了，但是向导必须用在固定处理流程中，并且处理流程应该不少于 3 个处理步骤。

7）用户帮助

系统应该提供详尽而可靠的帮助文档,在用户使用产生迷惑时可以自己寻求解决方法。

常用的帮助设施有两种: 集成的和附加的。集成的帮助设施一开始就是设计在软件中的，它与语境有关，用户可以直接选择与所要执行操作相关的主题。通过集成帮助设施可以缩短用户获得帮助的时间，增加界面的友好性，附加的帮助设施在系统建好以后再加进去，通常是一种查询能力比较弱的联机帮助。

（1）帮助文档中的性能介绍与说明要和系统性能配套一致；

（2）操作时要提供及时调用系统帮助的功能，常用 F1；

（3）最好提供目前流行的联机帮助格式或 HTML 帮助格式；

（4）用户可以用关键词在帮助索引中搜索所要的帮助，当然也应该提供帮助主题词；

（5）在帮助中应该提供我们的技术支持方式，一旦用户难以自己解决可以方便地寻求新的帮助方式。

8）出错信息和警告

出错信息和警告是指出现问题时系统给出的坏消息,信息以用户可以理解的术语描述。

（1）信息应提供如何从错误中恢复的建设性意见；

（2）信息应指出错误可能导致哪些不良后果，以便用户检查是否出现了这些情况并帮助用户进行改正；

（3）信息应伴随着视觉上的提示，如特殊的图像、颜色或者信息闪烁；

（4）信息不能带有判断色彩，即在任何情况下不能指责用户。

9）一般交互

（1）一致性：菜单选择、数据显示以及其它功能都应使用一致的格式；

（2）提供有意义的反馈；

（3）在数据录入上允许取消大多数操作；

（4）减少在动作间必须记忆的信息数量；

（5）允许用户非恶意错误，系统应保护自己不受致命错误的破坏。

10）数据输入

（1）尽量减少用户输入动作的数量；

（2）维护信息显示和数据输入的一致性；

（3）交互应该是灵活的，对键盘和鼠标输入的灵活性提供支持；

（4）在当前动作的语境中使不合适的命令不起作用。

11）独特性

如果一味地遵循业界的界面标准，则会丧失自己的个性。在框架符合规范的情况下，设计具有自己独特风格的界面尤为重要，在商业软件流通中会有很好的潜移默化的广告效用。安装界面上应有单位介绍或产品介绍，并有自己的图标。

第二章、数据库设计原则

数据库技术是信息资源管理最有效的手段。数据库设计是建立数据库及其应用系统的核心和基础，它要求对于指定的应用环境，构造出较优的数据库模式，建立起数据库应用系统，并使系统能有效地存储数据，满足用户的各种应用需求。

1、设计数据库之前

1）理解客户需求，询问用户如何看待未来需求变化。让客户解释其需求，而且随着开发的继续，还要经常询问客户以保证其需求仍然在开发的目的之中；

2）了解企业业务，在以后的开发阶段节约大量时间；

3）重视输入输出。在定义数据库表和字段需求（输入）时，首先应检查现有的或者已经设计出的报表、查询和视图（输出），以决定为了支持这些输出哪些是必要的表和字段；

4）创建数据字典和 E-R 图，对 SQL 表达式的文档化来说这是完全必要的；

5）定义标准的对象命名规范。

2、表与字段的设计

1）表设计原则

（1）标准化和规范化；

数据的标准化有助于消除数据库中的数据冗余。标准化有好几种形式，但 Third Normal Form（3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。事实上，为了效率的缘故，对表不进行标准化有时也是必要的。

（2）采用数据驱动，增强系统的灵活性与扩展性；

（3）在设计数据库的时候考虑到哪些数据字段将来可能会发生变更。

2）字段设计原则

（1）每个表中都应该添加的 3 个有用的字段；

① dRecordCreationDate，在 SQL Server 下默认为 GETDATE()；

② sRecordCreator，在 SQL Server 下默认为 NOT NULL DEFAULT USER；

③ nRecordVersion，记录的版本标记，有助于准确说明记录中出现 null 数据或者丢失数据的原因。

（2）对地址和电话采用多个字段，电话号码和邮件地址最好拥有自己的数据表，其间具有自身的类型和标记类别；

（3）使用角色实体定义属于某类别的列，创建特定的时间关联关系，从而可以实现自我文档化；

（4）选择数字类型和文本类型要尽量充足，否则无法进行计算操作；

（5）增加删除标记字段。在关系数据库里不要单独删除某一行，而在表中包含一个“删除标记”字段，这样就可以把行标记为删除。

3、键和索引

1）键选择原则

（1）键设计 4 原则

①所有的键都必须唯一；

②为关联字段创建外键；

③避免使用复合键；

④外键总是关联唯一的键字段。

（2）使用系统生成的主键，控制数据库的索引完整性，并且当拥有一致的键结构时，找到逻辑缺陷很容易；

（3）不要用用户的键，通常情况下不要选择用户可编辑的字段作为键；

（4）可选键有时可作主键，能拥有建立强大索引的能力。

2）索引使用原则

索引是从数据库中获取数据的最高效方式之一，绝大多数的数据库性能问题都可以采用索引技术得到解决。

（1）逻辑主键使用唯一的成组索引，对系统键（作为存储过程）采用唯一的非成组索引，对任何外键列采用非成组索引。考虑数据库的空间有多大，表如何进行访问，还有这些访问是否主要用于读写；

（2）大多数数据库都索引自动创建的主键字段，但是不能忘了索引外键，它们也是经常使用的键；

（3）不要索引 memo/note 字段，不要索引大型字段，这样会让索引占用太多的存储空间；

（4）不要索引常用的小型表，不要为小型数据表设置任何键，尤其当它们经常有插入和删除操作时。

4、数据完整性设计

1）完整性实现机制

（1）实体完整性：主键

（2）参照完整性

① 父表中删除数据：级联删除，受限删除，置空值；
② 父表中插入数据：受限插入，递归插入；
③ 父表中更新数据：级联更新，受限更新，置空值。

DBMS 对参照完整性可以有两种方法实现：外键实现机制（约束规则）和触发器实现机制。

（3）用户定义完整性：NOT NULL，CHECK，触发器。

2）用约束而非商务规则强制数据完整性；

3）强制指示完整性。在有害数据进入数据库之前将其剔除，激活数据库系统的指示完整性特性；

4）使用查找控制数据完整性，控制数据完整性的最佳方式就是限制用户的选择；

5）采用视图。可以为应用程序建立专门的视图而不必非要应用程序直接访问数据表，这样做还等于在处理数据库变更时给你提供了更多的自由。

5、其他设计

1）避免使用触发器，确实需要的话最好集中对它文档化；

2）使用常用英语（或者其他任何语言）而不要使用编码，确实需要的话可以在编码旁附上用户知道的英语；

3）保存常用信息。让一个表专门存放一般数据库信息，可以实现一种简单机制跟踪数据库，这样做对非客户机/服务器环境特别有用；

4）包含版本机制，在修改数据库结构时更为方便；

5）编制文档，对所有的快捷方式、命名规范、限制和函数都要编制文档；

6）反复测试，保证选择的数据类型满足商业要求；

7）检查设计，在开发期间检查数据库设计的常用技术是通过其所支持的应用程序原型检查数据库。

6、数据库命名规范

1）实体（表）的命名

（1）表以名词或名词短语命名，给表的别名定义简单规则；

（2）如果表或者是字段的名称仅有一个单词，那么建议不使用缩写，而是用完整的单词；

（3）所有的存储值列表的表前面加上前缀 Z，目的是将这些值列表类排序在数据库最后；

（4）所有的冗余类的命名(主要是累计表)前面加上前缀 X。冗余类是为了提高数据库效率，非规范化数据库的时候加入的字段或者表；

（5）关联类通过用下划线连接两个基本类之后，再加前缀 R 的方式命名，后面按照字母顺序罗列两个表名或者表名的缩写。关联表用于保存多对多关系。

2）属性（列）的命名

（1）采用有意义的列名，表内的列要针对键采用一整套设计规则；

每一个表都将有一个自动 ID 作为主健，逻辑上的主健作为第一组候选主健来定义。如果是自定义的逻辑上的编码则用缩写加“ID”的方法命名。如果键是数字类型，你可以用_NO 作为后缀。如果是字符类型则可以采用_CODE 后缀。对列名应该采用标准的前缀和后缀。

（2）所有的属性加上有关类型的后缀，如果还需要其它的后缀，都放在类型后缀之前。数据类型是文本的字段，类型后缀 TX 可以不写，有些类型比较明显的字段也可以不写类型后缀；

（3）采用前缀命名。给每个表的列名都采用统一的前缀，那么在编写 SQL 表达式的时候会得到大大的简化，但这样做也有缺点，比如会破坏自动表连接工具的作用。

3）视图的命名

（1）视图以 V 作为前缀，其他命名规则和表的命名类似；

（2）命名应尽量体现各视图的功能。

4）触发器的命名

触发器以 TR 作为前缀，触发器名为相应的表名加上后缀，Insert 触发器加 _I ，Delete 触发器加 _D ，Update 触发器加 _U 。如：TR_User_I，TR_User_D，TR_User_U。

5）存储过程名

存储过程应以 UP_ 开头，和系统的存储过程区分，后续部分主要以动宾形式构成，并用下划线分割各个组成部分。

6）变量名

变量名采用小写，若属于词组形式，用下划线分隔每个单词。

7）命名中其他注意事项

（1）以上命名都不得超过 30 个字符的系统限制，变量名的长度限制为 29（不包括标识字符@）；

（2）数据对象、变量的命名都采用英文字符，禁止使用中文命名，绝对不要在对象名的字符之间留空格；

（3）小心保留词，要保证你的字段名没有和保留词、数据库系统或者常用访问方法冲突；

（4）保持字段名和类型的一致性，在命名字段并为其指定数据类型的时候一定要保证一致性。

第三章、编程规范总则

1、排版

1）程序块要采用缩进风格编写，缩进的空格数为 4 个，对于由开发工具自动生成的代码可以不一致；

2）相对独立的程序块之间、变量说明之后必须加空行；

3）较长的语句要分成多行书写，长表达式要在低优先级操作符处划分新行操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读；

4）循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分，同 3）；

5）若函数或过程中的参数较长，也要进行适当的划分；

6）不允许把多个短语句写在一行中，即一行只写一条语句；

7）if、for、do、while、case、switch、default 等语句自占一行，且 if、for、do、while 等语句的执行语句部分无论多少都要加括号{}；

8）对齐只使用空格键，不使用 TAB 键；

9）函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case 语句下的处理语句也要遵从语句缩进要求；

10）程序块的分界符（如大括号‘{’和‘}’）应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始定义、类的定义、结构的定义、枚举的定义以及 if、for、do、while、switch、case 语句中的程序都要采用如上的缩进方式；

11）在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格，但不要连续留两个以上空格。进行非对等操作时，如果是关系密切的操作符（如－>）后不应加空格。

采用这种松散方式编写代码的目的是使代码更加清晰，在已经非常清晰的语句中没有必要再留空格。如果语句已足够清晰，则括号内侧(即左括号后面和右括号前面)不需要加空格，多重括号间也不必加空格。在长语句中，如果需要加的空格非常多，那么应该保持整体清晰，而在局部不加空格。

（1）逗号、分号只在后面加空格。

```
int a, b, c;
```

（2）比较操作符、赋值操作符、算术操作符、逻辑操作符、位操作符等双目操作符的前后加空格。

```
a = b c;
a *= 2;
a = b ^ 2;
```

（3）"!"、"~"、" "、"--"、"&"（地址运算符）等单目操作符前后不加空格。

```
flag = !isFull;
p = &com;
i ;
```

（4）"->"、"."前后不加空格。

（5）if、for、while、switch 等与后面的括号间应加空格，使 if 等关键字更为突出、明显。

```
if (a >= b && c > d)
```

12）一行程序以小于 80 字符为宜，不要写得过长。

2、注释

注释应该说明代码的目的，要讲清为什么要那么做，而不是怎么去做。

1）一般情况下，源程序有效注释量必须在 20％以上。注释的原则是有助于对程序的阅读理解，在该加的地方都加，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁；

2）注释格式尽量统一，建议使用“/* …… */”;

3）说明性文件（如头文件.h 文件、.inc 文件等）头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容

、功能、与其它文件的关系等，头文件的注释中还应有函数功能简要说明；

4）源文件头部应进行注释，列出：版权说明、版本号、生成日期、作者、模块功能、主要函数及其功能等；

5）函数头部应进行注释，列出：函数功能、输入参数、输出参数、返回值等；

6）边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一

致性；

7）避免在注释中使用缩写，特别是非常用的缩写。如无法避免，应对缩写进行必要的说明；

8）注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，如放于上方则需与其上面

的代码用空行隔开；

9）变量、常量、宏的注释有时也是必须的，应放在其上方相邻位置或右方；

10）数据结构声明(包括数组、结构、类、枚举等)，如果其命名不是充分自注释的，必须加以注释。对数据结构的注释应放在其上方相邻位置，对结构中的每个域的注释放在此域的右方；

11）全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明；

12）将注释与其上面的代码用空行隔开，注释与所描述内容进行同样的缩排；

13）对变量的定义和分支语句（条件分支、循环语句等）必须编写注释。这些语句往往是程序实现某一特定功能的关键，对于维

护人员来说，良好的注释帮助更好地理解程序，有时甚至优于看设计文档；

14）通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的，减少不必要的注释；

15）当代码段较长，特别是多重嵌套时，在程序块的结束行右方加注释标记，以表明某程序块的结束；

16）建议注释多使用中文，除非能用非常流利准确的英文表达。

3、标识符命名

1）标识符的命名要清晰明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。较长的单词可取单词的头几个字母形成缩写，一些单词有大家公认的缩写；

2）命名中若使用特殊约定或缩写，应该在源文件的开始之处，进行必要的注释说明；

3）命名风格要自始至终保持一致；

4）对于变量命名，禁止取单个字符（如 i、j、k...）。单个字符容易敲错，且编译时又不易检查出来。建议除了要有具体含义外，还能表明其变量类型、数据类型等，但 i、j、k 作局部循环变量是可以的；

5）命名规范必须与所使用的系统风格保持一致，并在同一项目中统一。除非

必要，不要用数字或较奇怪的字符来定义标识符；

6）在同一软件产品内，应规划好接口部分标识符（变量、结构、函数及常量）的命名，防止编译、链接时产生冲突；

7）用正确的反义词组命名具有互斥意义的变量或相反动作的函数等；

下面是一些在软件中常用的反义词组：

add / remove begin / end create / destroy
insert / delete add / delete get / release
increment / decrement put / get
lock / unlock open / close first / last
min / max old / new start / stop
next / previous send / receive show / hide
source / target source / destination
cut / paste up / down

8）除了特殊应用，应避免使用以下划线开始和结尾的定义。

4、可读性

1）注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级；

2）避免使用不易理解的数字，用有意义的标识来替代；

3）源程序中关系较为紧密的代码应尽可能相邻，便于程序阅读和查找；

4）不要使用难懂的技巧性很高的语句，除非很有必要时。程序的高效率并不等同于语句的高技巧，而在于算法。

5、变量与结构

1）去掉没必要的公共变量，以降低模块间的耦合度；

2）仔细定义并明确公共变量的含义、作用、取值范围及公共变量间的关系；

3）明确公共变量与操作此公共变量的函数或过程的关系，如访问、修改及创建等。这种关系的说明可在注释或文档中描述；

4）当向公共变量传递数据时，要十分小心，防止赋与不合理的值或越界等现象发生。若有必要应进行合法性检查，以提高代码的可靠性、稳定性；

5）构造仅有一个模块或函数可以修改、创建，而其余有关模块或函数只访问的公共变量，防止多个不同模块或函数都可以修改、创建同一公共变量的现象；

6）使用严格形式定义的、可移植的标准数据类型，尽量不要使用与具体硬件或软件环境关系密切的变量；

7）结构的功能要单一，是针对一种事务的抽象。结构中的各元素应代表同一事务的不同侧面，而不应把描述没有关系或关系很弱的不同事务的元素放到同一结构中；

8）不同结构间的关系不要过于复杂，否则应合为一个结构；

9）仔细设计结构中元素的布局与排列顺序，使结构容易理解、节省占用空间，并减少引起误用的现象；

10）结构的设计要尽量考虑向前兼容和以后的版本升级，并为某些未来可能的应用保留余地；

11）留心具体语言及编译器处理不同数据类型的原则及有关细节；

12）编程时，要注意数据类型的强制转换。对编译系统默认的数据类型转换要有充分的认识，尽量减少没有必要的数据类型默认转换与强制转换，合理地设计数据并使用自定义数据类型，避免数据间进行不必要的类型转换；

13）对自定义数据类型进行恰当命名，使它成为自描述性的，以提高代码可读性，但要注意其命名方式在同一产品中的统一。

6、函数与过程

1）设计高扇入、合理扇出（小于 7）的函数。较良好的软件结构通常是顶层函数的扇出较高，中层函数的扇出较少，而底层函数

则扇入到公共模块中；

2）函数的规模尽量限制在 200 行以内，不包括注释和空格行；

3）对所调用函数的错误返回码要仔细、全面地处理；

4）在同一项目组应明确规定对接口函数参数的合法性检查应由函数的调用者负责还是由接口函数本身负责，缺省是由函数调用

者负责；

5）防止将函数的参数作为工作变量。对必须改变的参数，最好先用局部变量代之，再将该局部变量的内容赋给该参数；

6）一个函数仅完成一件功能，不要设计多用途的函数。函数名应准确描述函数的功能；

7）函数的功能应该是可以预测的，也就是说只要输入数据相同就应产生同样的输出；

8）避免设计多参数函数，不使用的参数从接口中去掉，减少函数间接口的复杂度；

9）非调度函数应减少或防止控制参数，尽量只使用数据参数，防止函数间的控制耦合；

10）检查函数所有参数输入与非参数输入的有效性；

11）在编程时，经常遇到在不同函数中使用相同的代码，许多开发人员都愿把这些代码提出来，并构成一个新函数。若这些代码

关联较大并且是完成一个功能的，那么这种构造是合理的，否则这种构造将产生随机内聚的函数；

12）功能不明确且较小的函数，特别是仅有一个上级函数调用它时，应考虑把它合并到上级函数中，而不必单独存在；

13）减少函数本身或函数间的递归调用。除非为某些算法或功能的实现方便，应减少没必要的递归调用；

14）仔细分析模块的功能及性能需求，并进一步细分，若有必要画出有关数据流图，据此来进行模块的函数划分与组织；

15）对于提供了返回值的函数，在引用时最好使用其返回值；

16）当一个过程（函数）中对较长变量（一般是结构的成员）有较多引用时，可以用一个意义相当的宏代替。

7、可测性

1）在同一项目组或产品组内，要有一套统一的为集成测试与系统联调准备的调测开关及相应打印函数，并且要有详细的说明；

2）在同一项目组或产品组内，调测打印出的信息串的格式要有统一的形式。信息串中至少要有所在模块名（或源文件名）及行号；

3）编程的同时要为单元测试选择恰当的测试点，并仔细构造测试代码、测试用例，同时给出明确的注释说明。测试代码部分应作为（模块中的）一个子模块，以方便测试代码在模块中的安装与拆卸（通过调测开关）；

4）使用断言来发现软件问题，提高代码可测性。用断言来检查程序正常运行时不应发生但在调测时有可能发生的非法情况，但不能用断言来检查最终产品肯定会出现且必须处理的错误情况；

5）对较复杂的断言加上明确的注释，用断言确认函数的参数，保证没有定义的特性或功能不被使用，对程序开发环境的假设进行检查；

6）正式软件产品中应把断言及其它调测代码去掉（即把有关的调测开关关掉），以加快软件运行速度；

7）在软件系统中设置与取消有关测试手段，不能对软件实现的功能等产生影响；

8）用调测开关来切换软件的 DEBUG 版和正式版，而不要同时存在正式版本和 DEBUG 版本的不同源文件，以减少维护的难度；

9）软件的 DEBUG 版本和发行版本应该统一维护，不允许分家，并且要时刻注意保证两个版本在实现功能上的一致性；

10）在编写代码之前，应预先设计好程序调试与测试的方法和手段，并设计好各种调测开关及相应测试代码如打印函数等；

11）调测开关应分为不同级别和类型。针对模块或系统某部分代码的调测，针对模块或系统某功能的调测，对性能、容量等的测

试；

12）编写防错程序，然后在处理错误之后可用断言宣布发生错误。

8、程序效率

1）在保证软件系统的正确性、稳定性、可读性及可测性的前提下提高代码效率，包括全局效率、局部效率、时间效率及空间效率；

2）局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响；

3）通过对系统数据结构的划分与组织的改进，以及对程序算法的优化来提高空间效率；

4）仔细考虑循环体内的语句是否可以放在循环体之外，使循环体内工作量最小，从而提高程序的时间效率；

5）仔细考查、分析系统及模块处理输入（如事务、消息等）的方式，并加以改进；

6）对模块中函数的划分及组织方式进行分析、优化，改进模块中函数的组织结构，提高程序效率；

7）不应花过多的时间拼命地提高调用不很频繁的函数代码的效率；

8）仔细地构造或直接用汇编编写调用频繁或性能要求极高的函数。嵌入汇编可提高时间及空间效率，但也存在一定风险；

9）在保证程序质量的前提下，通过压缩代码量、去掉不必要代码以及减少不必要的局部和全局变量，来提高空间效率；

10）尽量减少循环嵌套层次。在多重循环中，应将最忙的循环放在最内层，以减少 CPU 切入循环层的次数；

11）避免循环体内含判断语句，应将循环语句置于判断语句的代码块之中；

12）尽量用乘法或其它方法代替除法，特别是浮点运算中的除法；

13）不要一味地追求紧凑的代码，因为紧凑的代码并不代表高效的机器码。

9、质量保证

1）在软件设计过程中构筑软件质量；

2）代码质量保证优先原则

（1）正确性，指程序要实现设计要求的功能；

（2）稳定性/安全性，指程序稳定、可靠、安全；

（3）可测试性，指程序要具有良好的可测试性；

（4）规范/可读性，指程序书写风格、命名规则等要符合规范；

（5）全局效率，指软件系统的整体效率；

（6）局部效率，指某个模块、子模块、函数的本身效率；

（7）个人表达方式，指个人编程习惯。

3）只引用属于自己的存贮空间；

4）防止引用已经释放的内存空间；

5）过程/函数中分配的内存，在过程/函数退出之前要释放；

6）过程/函数中申请的（为打开文件而使用的）文件句柄，在过程/函数退出之前要关闭；

7）防止内存操作越界；

8）认真处理程序所能遇到的各种出错情况；

9）系统运行之初，要初始化有关变量及运行环境，防止未经初始化的变量被引用，并对加载到系统中的数据进行一致性检查；

10）严禁随意更改其它模块或系统（不属于自己）的有关设置和配置，不能随意改变与其它模块的接口；

11）注意易混淆的操作符。当编完程序后，应从头至尾检查一遍这些操作符，以防止拼写错误；

12）有可能的话，if 语句尽量加上 else 分支，对没有 else 分支的语句要小心对待。switch 语句必须有 default 分支；

13）不使用与硬件或操作系统关系很大的语句，而使用建议的标准语句，以提高软件的可移植性和可重用性；

14）精心构造算法，并对其性能、效率进行测试，对较关键的算法最好使用其它算法来确认；

15）注意表达式是否会上溢、下溢，使用变量时要注意其边界值；

16）系统应具有一定的容错能力，对一些错误事件（如用户误操作等）能进

行自动补救；

17）对一些具有危险性的操作代码要仔细考虑，防止对数据、硬件等的安全构成危害，以提高系统的安全性。

10、代码编辑、编译与审查

1）同产品软件（项目组）内，最好使用相同的编辑器，并使用相同的设置选项；

2）打开编译器的所有告警开关对程序进行编译；

3）通过代码走读及审查方式对代码进行检查；

4）编写代码时要注意随时保存，并定期备份，防止由于断电、硬盘损坏等原因造成代码丢失；

5）某些语句经编译后产生告警，如果你认为它是正确的，那么应通过某种手段去掉告警信息；

6）使用代码检查工具对源程序检查，使用软件工具进行代码审查。

11、代码测试与维护

1）单元测试要求至少达到语句覆盖；

2）整理或优化后的代码要经过审查及测试；

3）代码版本升级要经过严格测试；

4）使用工具软件对代码版本进行维护；

5）正式版本上软件的任何修改都应有详细的文档记录；

6）发现错误立即修改，并且要记录下来；

7）关键的代码在汇编级跟踪；

8）仔细设计并分析测试用例，使测试用例覆盖尽可能多的情况，以提高测试用例的效率；

9）尽可能模拟出程序的各种出错情况，对出错处理代码进行充分的测试；

10）仔细测试代码处理数据、变量的边界情况；

11）保留测试信息，以便分析、总结经验及进行更充分的测试；

12）不应通过“试”来解决问题，应寻找问题的根本原因；

13）对自动消失的错误进行分析，搞清楚错误是如何消失的；

14）测试时应设法使很少发生的事件经常发生；

15）明确模块或函数处理哪些事件，并使它们经常发生；

16）坚持在编码阶段就对代码进行彻底的单元测试，不要等以后的测试工作来发现问题；

17）去除代码运行的随机性，让函数运行的结果可预测，并使出现的错误可再现。